INFORMS Journal on Computing
Vol. 10, No. 4, Fall 1998

0899-1499/98/1004-0384 $05.00

COMMENTARY

Dynamic, Distributed, Platform Independent OR/MS Applications—A Network Perspective

GORDON H. BRADLEY / *Operations Research Department, Naval Postgraduate School, Monterey, CA 93943; Email: bradley@nps.navy.mil*

ARNOLD H. BUSS / *Operations Research Department, Naval Postgraduate School, Monterey, CA 93943; Email: bussa@or.nps.navy.mil*

We are in the midst of, or perhaps at the beginning of, the development of a new concept of computing and computers that will significantly affect OR/MS. This development is summarized in the phrase "the network is the computer." The focus and use is not an individual computer and its resources but a network of computers and resources, indeed, a network of all computers. This concept is closely tied to an evolving view about what a computer is; we now have an exploding number of ubiquitous devices with computational capacity, such as credit cards, cellular phones, and TV set-top boxes. We need to anticipate the impact of millions and then billions of programmable devices that are interconnected on a high bandwidth network accessible by millions and then billions of individuals. The sudden and fast moving development of computer networks and the wide distribution of powerful computing devices is revolutionizing not just OR/MS, but commerce, communications, and indeed, nearly all aspects of society.

The feature article by Bhargava and Krishnan (BK) is an important contribution to an unfolding dialogue about how OR/MS will evolve and redefine itself as computer networks emerge as the environment of choice for many applications. The question is not just how existing products will be distributed and accessed on the network, but how OR/MS products and ideas will be used, what new products will be developed, and how OR/MS will be reshaped by the changing environment and the new concepts.

As its title indicates, the BK article is focused on the use of the Web and browsers rather than on the larger use of computer networks. This viewpoint is based on their experience with DecisionNet[1] that they and others developed over the last several years. DecisionNet is an inspired, well-developed application that uses the Web and browsers to effectively deliver OR/MS applications to a wider audience. The authors have developed a market-based approach that uses a software broker to connect producers with consumers of OR/MS applications. DecisionNet effectively achieves three objectives: (1) it makes applications visible to potential consumers, (2) it avoids moving (porting) applications to the (variety) of consumer computing environments, and (3) it makes it easy for consumers to submit data to the producer's computer.

There are OR/MS applications that use computer networks without necessarily using the Web or browsers. Since 1996, we have been working with graduate students to develop a number of network-based OR/MS applications. Although these applications operate over the Internet, most do not use the Web or a browser. We have found that Java applications (together with RMI or CORBA), rather than Java applets and other Web technologies, provide a better platform for building our network-based systems. Because we have specific problem domains with their own issues, we have different experiences and a somewhat different perspective and emphasis than that discussed in this feature article. We will give a short description of one of our systems and then relate our experience to the issues discussed in the BK article. This should be useful to others working on network applications.

1. Map-Based Planning on the Internet

The problem domains we address include military planning systems for contemporary and future needs that utilize the advanced information technologies that will be widely available in the early part of the next century. High-level Department of Defense long-range studies, such as Joint Vision 2010, New World Vistas, Army XXI, and Army After Next, have specified a set of requirements for these planning systems that cannot be achieved with existing technologies. The post Cold War era is characterized by a wider range of possible military missions than the Cold War era, from full-scale war to small peacekeeping missions, and by greater uncertainty about when and where military response will be required. This has generated requirements for flexible planning tools that support rapid response to situations whose details cannot be anticipated. These requirements are extensive. The planning resources (people, data, computers) will be distributed over a network that has a range of computers (from supercomputers to cellular phones) and different software (operating systems, databases). The decision cycles will be much shorter, meaning the planning systems need to work much faster with less time to make and review plans. Furthermore, there is a requirement to provide automatic monitoring of the planning process. The problems faced by planners will be less predictable than in the past, so the systems must be more flexible to address situations the designers cannot anticipate. The systems must have an open architecture that allows additional capabilities to be added without disruption. Leg-

acy planning systems are too static, monolithic, and inflexible to meet these requirements. Current efforts to integrate legacy planning tools are an improvement, but even when these efforts are brought to fruition, the results will not be sufficiently interoperable, platform independent, or extensible to meet the challenges of military decision making outlined above. As demanding as the individual requirements are, advanced planning systems must incorporate all these capabilities in an integrated system.

We will describe our most recent application that was done with Arent Arntzen, Allan Bilyeu, and Leroy Jackson and is described in Bilyeu.[2] The system is demonstrated with a particularly demanding scenario of a special forces operation that involves Air Force units on the ground in one country, Naval units on a ship, and Army units in a second country working with UN forces from several different countries. The planning must be coordinated among all the forces and with other government agencies, and it is not possible to do the planning in a single location.

We chose Java as the programming language and implemented the required components as Java applications. Unlike Java applets that execute within a browser, Java applications execute as processes on the computer like any other application. The Internet protocols are embedded in the Java language itself, so content (for example, data, images) can be loaded from either local resources or over the Internet in a way that is relatively seamless. A Java application can do anything a Java applet can (load an image, invoke a common gateway interface script, etc.), but an application does not require a security policy as restrictive as that imposed on applets. The next version of Java, JDK 1.2, will ease the security limitations placed on applets. However, because applets are downloaded over the Web and executed without any action by the viewer, strict security measures are necessary to protect the viewer's computer from harmful actions.

Our application supports map-based military planning using network algorithms. The networks are created on the fly from disparate sources of data. Briefly, our system loads images (maps and satellite pictures), overlays (containing networks), and algorithms that operate on the networks in the overlays. The dynamic capabilities of the system are best shown by considering how algorithms are located, loaded, and executed. The user selects one of the networks embedded in an overlay and then selects an algorithm (for example, shortest path or max flow) to execute on it. The algorithms are not preloaded; because Java has dynamic loading, classes are not loaded until they are first used. A class containing an algorithm can be loaded from the local machine or over the Internet. Once selected, the system loads the class and, using the name of the class (a string) and a process called reflection, determines the names of the algorithms in the class and presents them to the planner. After the planner selects an algorithm, the system uses reflection to determine the parameters needed to execute the algorithm (for example, for a shortest path algorithm, the parameters are a source node and an arc property for the length). The system presents the planner with a short description of the algorithm and drop-down menus to select which of the nodes is the source and which arc property should be used for the length (there could be several choices, for example, length via road or length via air). The system then executes the algorithm and places the solution in the network. Because the algorithm is loaded dynamically, it need not be known at compile time or even at the time the planning session is begun. In an extreme example, an analyst at a remote location can be writing an algorithm while the planning is going on and, as soon as it is compiled, it can be located, loaded into the planning system, and executed on a network. The general capabilities for dynamic behavior are built into Java, but it should be noted that some specific capabilities are achieved only by using the loosely coupled components architecture that we have developed over the past year (see Bradley and Buss[3]).

2. Distribution Technologies—RMI and CORBA

The network can be used to distribute work, for example, sending data to remote algorithms, sending algorithms to remote data, invoking remote algorithms on remote data, invoking a method on a remote object, sending data and algorithms to a remote computer for execution, etc. The explosion of the Internet has helped spawn a number of supporting technologies that are inconceivable in the world of monolithic mainframe computers and stand-alone desktop computers. Several competing infrastructures have arisen for supporting distributed computing, including CORBA and Java's remote method invocation (RMI). Although we have used both CORBA and RMI, it appears that RMI ultimately offers more to support our applications and designs than does CORBA.

From its early, limited form, Java's RMI has blossomed into a rich, full-featured platform for distributed computing. RMI has most of the features that make CORBA attractive: an interface definition language (Java's API itself), a registry that performs the same functions as CORBA's object manager and naming service, support for both static remote binding via stubs and skeletons, and dynamic remote binding and class loading. RMI also supports such features as remote garbage collection and threads. Although not as full-featured as CORBA, RMI is sufficiently advanced to support most tasks required by the loosely coupled components architecture. Furthermore, there is a tremendous advantage in the fact that the API for remote objects under RMI is essentially the Java API as well. This avoids writing applications in two languages, as in CORBA (one being CORBA's interface definition language and the other being the implementation language). RMI has a registry for the naming and discovery of remote objects that provides the same functionality to clients as CORBA's object manager and naming service.

Java's serialization support, combined with RMI, enables objects to be saved in binary form, discovered and downloaded by an RMI client, and finally unpacked and used. This ability is particularly useful for distributed planning and analysis. For example, it will be possible for one analyst to formulate a network model using many disparate sources of data, then serialize just that network model and send it to

another analyst, who could be physically located anywhere. The second analyst could deserialize the network to exactly the same state as it was sent, after which any algorithms could be run on the network.

3. Why Java?

Why have we used Java (with some CORBA) rather than any of the other technologies discussed in BK? The first answer is a practical one—we wanted to limit the scope of the technology that we and our team had to master. Java is object-oriented, platform independent, and network aware. It also supports dynamic and distributed computation, ingredients that deliver precisely what we need in our problem domain of real-time military planning.

The second answer is that all our graduate students know Java and, through their classroom work, are familiar with object-oriented design and the power of Java and the network. Beginning with the students that entered in the fall of 1996, all master's students in the Operations Research Curriculum at the Naval Postgraduate School learn Java and use it in subsequent courses. Because we are educating military officers for a high-tech information age career, we view the move to the Internet and Java, "the language of the Internet," as a strategic curriculum decision to incorporate the use of networks. Just as important, it is possible to effectively introduce Java and object-oriented programming to operations research graduate students, many of whom have a very limited programming background, in a single course. Before Java was available, we had considered C++ and had decided that the cost to teach C++ to all our students was far greater than the benefits in the curriculum. With Java's cleaner design, the cost is far less, and the benefits of using a good object-oriented language and the Internet are significant. All master's students (input over 80 per year) write a thesis and many have used Java in their research.

The third answer is that the scripting languages associated with the Web and the security restrictions placed on Java applets limit the ability of OR/MS applications to scale-up to the size that many applications require. For the problem domains in which we work, it is not sufficient to just demonstrate the technology; we want to design, develop, and validate an architecture for network-based OR/MS that will support industrial size and industrial quality applications.

The final and most important reason for Java is that it contains powerful support for network operations, reflection, and dynamic loading. These are necessary for constructing the loosely coupled components architecture for our problem domains. As the first major language designed since the Web was introduced, Java has unique capabilities to support our work.

Most of our applications do not use the Web; however, the ones that do show its unique power. Tim Castle and Allan Washburn are working with us on an application to support land-based search and rescue (for lost people). They have built a Markov model of the movement of the individual that includes a Bayesian update based on the results of the search parties. We have superimposed a color display of the probability map over geographic maps to give a visual display of the state of the search. This application is implemented as an applet that is displayed in a browser and broadcast over the Web. This allows anyone (searchers, news media, and family) with a Web connection and a browser to view the same up-to-date picture that the search coordinators use.

4. Experience and Perspective

As discussed, most of our experience is in using the Internet, rather than the Web or browsers. Much, but not all, of the BK article is based on using the Web (the discussion summarized in BK's Table II is an exception because it includes Java applications and RMI). Our experience is heavily influenced by our problem domain. We are building applications for planning in which the OR/MS analyst is integrated into a dynamic, distributed system that must make a sequence of decisions in a rapidly changing and unpredictable environment. To contribute to the decision-making process, the analyst must, in a timely manner, use resources of various kinds that are spread over a computer network.

The efficiency of Java applications and applets has improved dramatically with the introduction of just-in-time compilers into the Java virtual machines, both in browsers and outside of browsers. The usual measure is to compare performance to C++ code. Current benchmarks show that Java can be several times slower, although one shows Java dead even for some very special numerical calculations. We have done no systematic testing, but we estimate (JDK 1.1, summer 1998) that Java execution is two times slower than C++ with a large variance. HotSpot technology, currently being developed by Sun Microsystems, has the potential to significantly speed-up Java. The most innovative part of the HotSpot technology is an adaptive compiler that creates a profile of the program during execution and recompiles the program while it is running to relieve bottlenecks (see [4] for several other ways that are being used to speed up Java execution). From our perspective, anything from slightly faster than C++ to several times slower is acceptable because the notion of time and efficiency in our system is "can the analysis be completed before the next decision is made." Meeting this requirement has less to do with how fast a numerical algorithm can execute on a specific CPU than with how quickly the algorithm itself can be identified, brought into the analysis, and loaded with data. If calculation speed is important, finding a faster CPU on the network is as effective as finding a faster program or algorithm.

From our perspective, Java applications with RMI (and CORBA, if needed) dominate approaches that use the Web and its associated technologies (for example, scripting languages) because Web technologies have limited capabilities that do not scale-up well. It appears to us that Java applications and RMI offer sufficient dynamic discovery capabilities. Java is adding CORBA capabilities (for example, interface definition language in Java) at a rate that may overtake Object Management Group, the consortium of over 700 companies that manages CORBA. In addition, Java support of Structural Query Language queries and Java wrappers for

programs written in other languages may limit our need for other technologies such as CORBA.

5. Final Comments

One of the benefits of a wide-angle overview of an emerging research field is that it gives other researchers an opportunity to reassess the assumptions that lead to specific design decisions. We used the work of BK for exactly that purpose in the summer of 1997 (with an early version of the article) and in the summer of 1998 (with this version). In both instances, we were able to use their careful review of the various options quickly to determine that, for our particular problem domain, we have made the right choices among the current alternatives.

The article by BK is provocative because it gives an OR/MS perspective on the network and the Web and shows and suggests and hints at the many ways that OR/MS will be reshaped and reconceived as we use the new capabilities and absorb the new ideas. Our work demonstrates the power of the ideas—by using Java applications rather than applets, we do not use the Web or browsers directly; but it was the idea and example of the Web (and the artifacts it spawned, such as Java) that made our work conceivable as well as possible.

Acknowledgments

The research of the authors was supported by grants from the Air Force Office of Scientific Research and the Office of Naval Research. This support is gratefully acknowledged.

References

1. H.K. BHARGAVA, R. KRISHNAN, and R. MULLER, 1997. Decision Support on Demand: Emerging Electronic Markets for Decision Technologies, *Decision Support Systems 19*, 193–214.
2. A.L. BILYEU, 1998. Concept for a Special Operations Planning and Analysis System, MS thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA.
3. G.H. BRADLEY and A.H. BUSS, 1998. An Architecture for Dynamic Planning Systems using Loosely Coupled Components, Technical Report NPS-OR-98-05, Naval Postgraduate School, Monterey, CA.
4. T.R. HALFHILL, 1998. How to Soup Up Java, *BYTE Magazine 23(5),* 60–74.